UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/696,867	10/30/2003	Conor Morrison	M1103.70436US00	8068

45840 7590 07/06/2009

WOLF GREENFIELD (Microsoft Corporation)
C/O WOLF, GREENFIELD & SACKS, P.C.
600 ATLANTIC AVENUE
BOSTON, MA 02210-2206

EXAMINER	
PESIN, BORIS M	

ART UNIT	PAPER NUMBER
2174	

MAIL DATE	DELIVERY MODE
07/06/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/696,867	MORRISON ET AL.	
	Examiner	Art Unit	
	BORIS PESIN	2174	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 December 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-11,13-23,26 and 27* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-11, 13-23 and 26-27* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Response to Amendment

This communication is responsive to the amendment filed 12/17/2008.

Claims 1-11, 13-23 and 26-27 are pending in this application. Claims 1, 15, and 26 are independent claims. In the amendment filed 12/17/2008 Claims 1-5, 15-17, and 26 were amended. This action is made Final.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 1 and 26-27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel ("Ezekiel"; US #5625783) in view of "Using adapters to reduce interaction complexity in reusable component-based software development" ("Rine", 1999) and further in view of Dobronsky et al. (Dobronsky, US 6784900).

As per independent claim 1, Ezekiel teaches a computer-implemented method of generating a componentized user interface for one or more computer applications, at least one of which is capable of performing one or more functions, the method comprising:

displaying a first set of interface elements provided by a framework (col 9, paragraph 4 where common commands included as placeholders is interpreted as a set of interface elements);

displaying a second set of interface elements provided by a first plug-in that is linked to the framework (abstract, add-on software components provides additional menu items is interpreted as one or more sets of interface elements) with a first plug-in that is linked to the framework (in fig 2, item 271 where add-on dll is interpreted as a plug-in);

displaying a third set of interface elements provided by a second plug-in that is linked to the framework (abstract and column 6 paragraph 2 add-on software components provides additional menu items is interpreted as one or more sets of interface elements) with a second plug-in that is linked to the framework (in fig 2, item 272 where add-on dll is interpreted as a plug-in).

Although Ezekiel shows providing an interface between one or more applications and the first plug-in and between the one or more applications and the second plug-in (fig 2 link between 260 and 271, 272). Ezekiel does not expressly disclose the link is a shell adapter, in order to utilize the second set of interface elements and the third set of

interface elements, wherein the shell adapter maps interface elements of the first plug-in and interface elements of the second plug-in to functions of one or more applications.

Ezekiel does not expressly disclose the link is a shell adapter, in order to utilize the second set of interface elements and the third set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins adapters are interpreted as one or more shell adapter interfaces).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine adapter in Ezekiel's method. The motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

Ezekiel as modified does not teach wherein the shell adapter maps functions of the first plug-in and functions of the second plug-in to functions of the one or more computer applications; and in response to a user activating an interface element provided by the first plug-in or the second plug-in, causing a computer application of the one or more computer applications to perform a function, where the shell adapter maps the interface element to the function. However, Dobronsky does teach this (Dobronsky Abstract, plugs-ins are used in the browser to allow for greater functionality. For example the plug-ins provide for other links which then use the browser's display functionality to display the appropriate link and its contents in the browser).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Dobronsky plug-ins with mappings to browser

functionality in the Ezekiel as modified method. The motivation would have been to be provide the user with greater core reusability and less memory consumption since only necessary plug-ins would need to be downloaded.

Claims 26 and 27 are similar in scope to claim 1; therefore they are rejected under similar rationale. Even though claim 26 recites a first application and a second application, those applications are not related or linked. Thus, the prior art still applies since you can have two separate instance of the same application running on two different computers.

Claims 2-5, 8-11, and 13 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine, and Dobronsky as applied to claim 1 above, and further in view of "Defining menus and toolbars in XML" ("Gehrman", Aug 2, 2002).

As per claim 2, Ezekiel and Rine teach the computer-implemented method of claim 1, wherein the first plug-in comprises:

(i) a first file that provides an interface between the framework and the first plug-in; and (ii) a second file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file).

Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (Introduction, paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing

in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML in Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 3, although Ezekiel does teach the plug-in contains menu items (Ezekiel, abstract) Gehrman further teaches these items are included in the XML file. (Gehrman Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file includes menu bars and toolbars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and tool bars with Ezekiel's method. The motivation is to facilitate customization of menus including menu bars and tool bars without changing the application's source code (Gehrman, introduction).

As per claim 4, Ezekiel and Rine teach the computer-implemented method of claim 1, wherein the second plug-in comprises:

(i) a first file that provides an interface between the framework and the second

plug-in; and (ii) a second file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file).

Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML file in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 5, although Ezekiel does teach the plug-in contains menu items (Ezekiel, abstract). Gehrman further teaches these items are included in the XML file. (Gehrman, Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file include menu bars and tool bars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and toolbars with Ezekiel's method. The motivation is to facilitate customization of menus including menu bars and toolbars without changing the application's source code (Gehrman, introduction).

As per claim 8, Ezekiel and Rine teach the computer-implemented method of claim 2, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises information written in an extensible markup language (XML).

However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 9, Ezekiel and Rine teach the computer-implemented method of claim 2, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises information written in a standard generalized markup language (SGML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is

interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 10, Ezekiel and Rine teach the computer-implemented method of claim 4, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises information written in an extensible markup language (XML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 11, Ezekiel and Rine teach the computer-implemented method of claim 4, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises a standard generalized markup language (SGML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 13, Ezekiel teaches the computer implemented method of claim 1, wherein the second set and the third set of interface elements comprise interface elements for the same application (Ezekiel, fig 2 items 260, 271, 272).

Claims 6, 7, and 14 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine and Dobronsky as applied to claim 1 above, and further in view of "Microsoft Office 2000/ Visual Basic: Programmer's Guide" ("Shank", April, 1999).

As per claim 6, Ezekiel and Rine teach the computer-implemented method of claim 1. Ezekiel and Rine do not teach wherein the framework is configured to discover the first plug-in and the second plug-in. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the default folder c:\windows\application Data\Microsoft\Addins is interpreted as the default folder the framework is setup to discover plug-ins).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder in Ezekiel's modified method. The motivation would have been to make it easier to locate an add-in (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 7).

As per claim 7, Even though Ezekiel shows the application locating and loading the plug-ins(fig 2, 260, 271, 272). He does not explicitly teach this is an automatic loading process. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader, it is interpreted in the broadest sense – an automatic loading of a plug-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's component loader in Ezekiel's modified method. The motivation would have been that the users do not have to browse to find

correct files. (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

As per claim 14, Ezekiel and Rine teach the computer-implemented method of claim 1 wherein the second set of interface elements comprises interface elements for a first application (Ezekiel abstract and fig 2 items 260, 271). Ezekiel and Rine do not explicitly teach the third set of interface elements comprise interface elements for a second application that is different from the first application. However Shank does teach this (Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where COM add-ins work in more than one of the applications).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's non-application specific add-in in Ezekiel's method. The motivation would have been to eliminate redundancy and be able to share code in the add-in (Shank, Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3).

Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel ("Ezekiel"; US #5625783) in view of "Using adapters to reduce interaction complexity in reusable component-based software development"("Rine", 1999) and "Microsoft Office 2000/ Visual Basic: Programmer's Guide" ("Shank", April, 1999) and further in view of Dobronsky et al. (Dobronsky, US 6784900).

As per independent claim 15, Ezekiel teaches a computer implemented method of providing extensibility to a user interface for one or more computer applications, at

least one of which is capable of performing one or more functions, the method comprising:

providing a framework the framework comprising a first set of interface elements (Ezekiel, col 9, paragraph 4 where *common commands included as placeholders* is interpreted as a set of interface elements). Although Ezekiel shows the locating and loading of the plug-ins(fig 2, 260, 271, 272), Ezekiel does not explicitly teach the addition of a user interface component loader, the framework configured to discover a plug-in located in a plug-in directory. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the default folder c:\windows\application Data\Microsoft\Addins is interpreted as the default folder the framework is setup to discover plug-ins) and (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader it is interpreted in the broadest sense – an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder and component loader in Ezekiel's method. The motivation would have been that it is easier to locate an add-in (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 7) and users do not have to browse to find correct files(Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

Ezekiel teaches loading the plug-in, the plug-in to provide a second set of interface elements. (Ezekiel abstract and col 6 par 2, *add-on software components provide additional menu items* are interpreted as one or more sets of interface elements). Ezekiel does not teach the plug-in was loaded with a user interface component loader. Shank does (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader it is interpreted in the broadest sense –an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder and component loader in Ezekiel's method. The motivation would have been that users do not have to browse to find correct files(Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

(d) Ezekiel teaches providing an interface between the one or more applications and the plug-in (fig 2 link between 260 and 271). Ezekiel and Shank do not expressly disclose the link is a shell adapter, in order to utilize the second set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins and adapter is interpreted as the shell adapter).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine's adapter in Ezekiel's modified method. The

motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

Ezekiel as modified does not teach wherein the shell adapter maps interface elements of the plug-in to functions of the one or more computer applications; and in response to a user activating an interface element provided by the plug-in, causing a computer application of the one or more computer applications to perform a function, where the shell adapter maps the interface element to the function. However, Dobronsky does teach this (Dobronsky Abstract, plugs-ins are used in the browser to allow for greater functionality. For example the plug-ins provide for other links which then use the browser's display functionality to display the appropriate link and its contents in the browser).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Dobronsky plug-ins with mappings to browser functionality in the Ezekiel as modified method. The motivation would have been to be provide the user with greater core reusability and less memory consumption since only necessary plug-ins would need to be downloaded.

Claims 16-23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine, Shank and Dobronsky as applied to claim 15 above, and further in view of "Defining menus and toolbars in XML" ("Gehrman", Aug 2, 2002).

As per claim 16, Ezekiel, Rine and Shank teach the computer-implemented method of claim 15, wherein the plug-in comprises:

(i) a first file that provides an interface between the framework and the plug-in; (ii) a second file that is written in a markup language and that includes menu elements (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (Introduction, paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML in Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 17, Ezekiel, Rine, Shank in view of Gehrman teach the computer-implemented method of claim 16, wherein the menu elements are selected from a group comprising of a toolbar, a status bar, and a menu bar. Although Ezekiel does teach the plug-in contains menu items (Ezekiel, abstract) Gehrman further teaches these items are included in the XML file. (Gehrman Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file include menu bars and toolbars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and toolbars with

Ezekiel's method. The motivation is to facilitate customization of menus including menu bars and toolbars without changing the application's source code (Gehrman, introduction).

As per claim 18, Ezekiel, Rine, Shank and Gehrman teach the computer-implemented method of claim 16, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises information written in an extensible markup language (XML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 19, Ezekiel, Rine, Shank and Gehrman teach the computer-implemented method of claim 16, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second

file comprises information written in a standard generalized markup language (SGML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 20, Ezekiel, Rine, Shank and Gehrman teach the computer implemented method of claim 15, wherein the framework is configured to provide the first set of interface elements (col 9, paragraph 4 where *common commands included as placeholders* is interpreted as a set of interface elements); Ezekiel does not teach the set is for a plurality of applications. However Gehrman does teach this (Gehrman page 3, paragraph 2 where standard actions are stored in a non-application specific class.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use Gehrman's separation of a set of standard actions, which are non-application specific. The motivation is to have the standard actions automatically be included in the application(s) without having to rewrite code. (Gehrman, page 3 paragraph 2).

As per claim 21, Ezekiel, Rine, and Shank teach the computer-implemented method of claim 15, wherein the method further comprises:

Ezekiel shows loading a second plug-in (Ezekiel fig 2, 260, 271, 272) providing a third set of interface elements (Ezekiel, abstract). Ezekiel does not explicitly teach the addition of a user interface component loader. However Shank does teach this (Shank chapter 2, “Deploying Templates and Application-Specific Add-ins” paragraph 3 where add-ins interpreted as plug-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader it is interpreted in the broadest sense –an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank’s default folder and component loader in Ezekiel’s method. The motivation would have been that users do not have to browse to find correct files(Shank, chapter 2, “Deploying Templates and Application-Specific Add-ins” paragraph 5).

Ezekiel teaches providing an interface between one or more applications and the second plug-in with a second shell interface in order to utilize the third set of interface elements. (fig 2 link between 260 and 271, 272). Ezekiel does not expressly disclose the link is a shell adapter interface, in order to utilize the third set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins and adapters are interpreted as one or more shell adapter interfaces).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine adapter in Ezekiel's method. The motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

As per claim 22, Ezekiel teaches the computer-implemented method of claim 21, wherein both the second set and the third set of interface elements comprise interface elements for a first application (Ezekiel, fig 2 items 260, 271, 272).

As per claim 23, Ezekiel, Rine, and Shank teach the computer-implemented method of claim 21 wherein the second set of interface elements comprises interface elements for a first application (Ezekiel fig2 items 260, 271). Ezekiel does not explicitly teach the third set of interface elements comprise interface elements for a second application that is different from the first application. However Shank does teach this (Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where COM add-ins work in more than one of the applications).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's non-application specific add-in in Ezekiel's method. The motivation would have been to eliminate redundancy and be able to share code in the add-in (Shank, Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where add-in is interpreted as plug-in).

Response to Arguments

Applicant's arguments with respect to claims 1-11, 13-23 and 26-27 have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Inquiry

Any inquiry concerning this communication or earlier communications from the examiner should be directed to BORIS PESIN whose telephone number is (571)272-4070. The examiner can normally be reached on Monday-Friday except every other Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Dennis Chow can be reached on (571)272-7767. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Boris Pesin/
Primary Examiner, Art Unit 2174